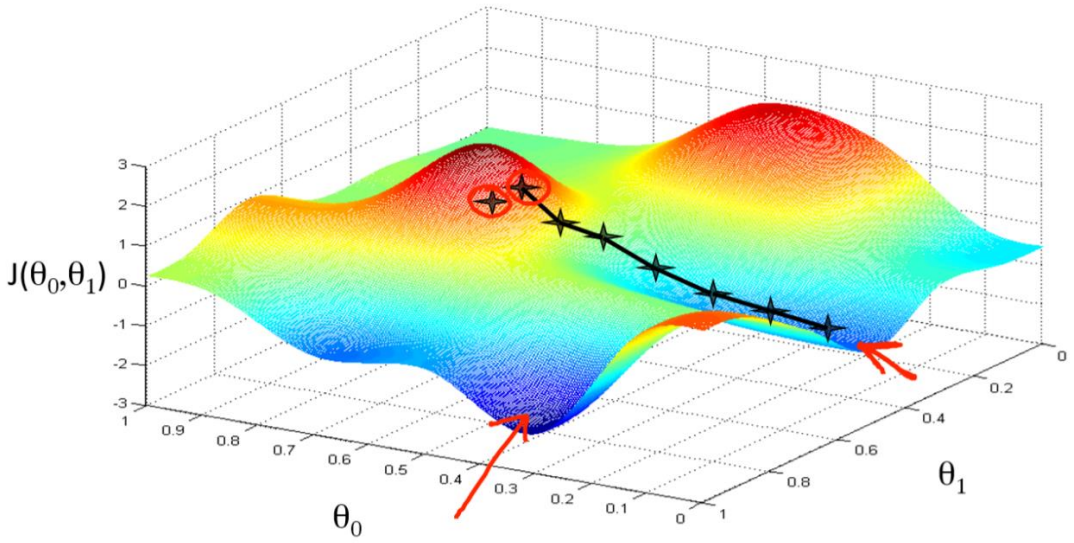

ASSIGNMENT 2

LEARNING A SIMPLE DEEP FORWARD NETWORK

Narayut Pudjaika (62070800407) | CPE 663 Deep Learning | 5 October 2020

Introduction

In this assignment, we will implement a backpropagation algorithm to learn a simple deep forward network for XOR function. Unlike the lecture which the XOR network uses sigmoid as the activation function. In this assignment we will use ReLU for the hidden units and sigmoid for the output unit. These are the architectural specification.

Architectural Specification

Connection type:	Fully connected
Number of input units (p):	2
Number of hidden layers:	1
Number of units in hidden layer (n):	2
Activation function of hidden layer:	ReLU
Number of output unit (m):	1
Activation function of output layer:	Sigmoid

TASKS

1. Draw the diagram showing this deep forward network.

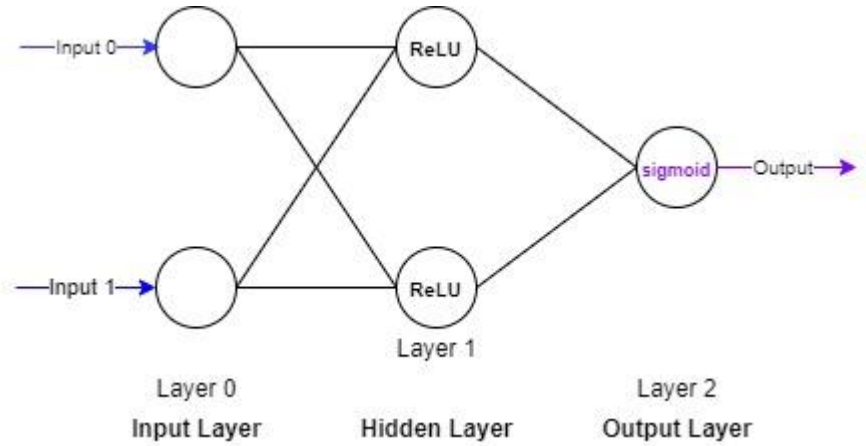

Figure 1 The picture of neural networks from the architectural specifications.

2. Write the equation of each units, showing all the components in the network.

Each input $x \in X$ is an input feeding into the networks. In each layer i which can be noticed by the superscript number according to a number of layers consists of its own parameters $W^{(i)}$ and activation function $g^{(i)}$ and bias $b^{(i)}$, the equations for each layer can be defined as.

Input Layer

$$\boldsymbol{x = a^{(0)}}$$

Hidden Layer (Layer 1: using ReLU as an activation function)

$$\boldsymbol{\sigma(z) = max\{0, z\}}$$

$$\boldsymbol{z^{(1)} = x^{T} W^{(1)} + b^{(1)}}$$

$$\boldsymbol{a^{(1)} = \sigma(z^{(1)})}$$

Output Layer (Layer 2: using Sigmoid as an activation function)

$$\boldsymbol{\sigma(z) = \frac{1}{1 + e^{-z}}}$$

$$\boldsymbol{z^{(2)} = a^{(1)T} W^{(2)} + b^{(2)}}$$

$$\boldsymbol{a}^{(2)} = \boldsymbol{\sigma}(\boldsymbol{z}^{(2)})$$

The chain rule can be expressed as.

$$f\left(x; W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\right) = \frac{1}{1 + e^{-\left(max\{0, x^{\mathrm{T}} W^{(1)} + b^{(1)}\}^{\mathrm{T}} W^{(2)} + b^{(2)}\right)}}$$

3. Write the error function.

Output Layer

$$E_{total}^{(2)} = -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} y_i \sigma(z_i^{(2)}) - (1 - y_i)\, log\left(1 - \sigma(z_i^{(2)})\right)$$

Hidden Layer

$$E_{total}^{(1)} = -\frac{1}{p} \sum_{k=1}^{p} \sum_{j=1}^{n} \sum_{i=1}^{m} \left(E_i^{(2)}\, w_{ij}^{(2)} \right) \sigma(z_j^{(1)})$$

Where **n** is the number of units in Hidden layer and **m** is the number of units in Output layer and **p** is the number of units in Input layer.

4. Derive the gradient of weights in output layers. Show your work.

$$\begin{aligned}
\frac{\partial E^{(2)}}{\partial w_{ij}^{(2)}} &= -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{y_i}{\sigma\left(z_i^{(2)}\right)} - \frac{(1 - y_i)}{1 - \sigma\left(z_i^{(2)}\right)} \frac{\partial \sigma}{\partial w_{ij}^{(2)}} \\
&= -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{y_i}{\sigma\left(z_i^{(2)}\right)} - \frac{(1 - y_i)}{1 - \sigma\left(z_i^{(2)}\right)} \frac{\partial \sigma}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial w_{ij}^{(2)}} \\
&= -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{y_i}{\sigma\left(z_i^{(2)}\right)} - \frac{(1 - y_i)}{1 - \sigma\left(z_i^{(2)}\right)} \sigma'\left(z_i^{(2)}\right) a_j^{(1)} \\
&= -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{y_i - y_i \sigma\left(z_i^{(2)}\right) - \sigma\left(z_i^{(2)}\right) + y_i \sigma\left(z_i^{(2)}\right)}{\sigma\left(z_i^{(2)}\right)\left(1 - \sigma\left(z_i^{(2)}\right)\right)} \sigma'(z_i^{(2)}) a_j^{(1)} \\
&= -\frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{\sigma'\left(z_i^{(2)}\right) a_j^{(1)}}{\sigma\left(z_i^{(2)}\right)\left(1 - \sigma\left(z_i^{(2)}\right)\right)} \left(\sigma\left(z_i^{(2)}\right) - y_i\right)
\end{aligned}$$

With sigmoid activation function, the derivative of $\sigma'\left(z_i^{(2)}\right)$ can be given by.

$$\sigma'\left(z_i^{(2)}\right) = \sigma\left(z_i^{(2)}\right)\left(1-\sigma\left(z_i^{(2)}\right)\right)$$

Substitute $\sigma'\left(z_i^{(2)}\right)$ and the gradient of the Output layer will be obtained.

$$\frac{\partial E^{(2)}}{\partial w_{ij}^{(2)}} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{m} a_j^{(1)}\left(\sigma\left(z_i^{(2)}\right)-y_i\right)$$

$$\frac{\partial E^{(2)}}{\partial w_{ij}^{(2)}} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{m} a_j^{(1)}\left(\left(\frac{1}{1+e^{-z_i^{(2)}}}\right)-y_i\right)$$

In a similar way, we can compute the partial derivative for the bias as.

$$\frac{\partial E^{(2)}}{\partial b_{ij}^{(2)}} = \frac{\partial E^{(2)}}{\partial z_i^{(2)}}\frac{\partial z_i^{(2)}}{\partial b_{ij}^{(2)}} = \frac{\partial E^{(2)}}{\partial z_i^{(2)}}(1) = \delta_i^{(2)} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{m}\left(\left(\frac{1}{1+e^{-z_i^{(2)}}}\right)-y_i\right)$$

5. Derive the gradient of weights in hidden layers. Show your work.

$$\begin{aligned}
\frac{\partial E^{(1)}}{\partial w_{jk}^{(1)}} &= -\frac{1}{p}\sum_{k=1}^{p}\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{\partial\left(E_i^{(2)}\,w_{ij}^{(2)}\right)}{\partial\sigma}\frac{\partial\sigma}{\partial z_j^{(1)}}\frac{\partial z_j^{(1)}}{\partial w_{jk}^{(1)}} \\
&= -\frac{1}{p}\sum_{k=1}^{p}\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{\partial E_i^{(2)}}{\partial\sigma}\frac{\partial\sigma}{\partial z_j^{(1)}}\frac{\partial z_j^{(1)}}{\partial w_{jk}^{(1)}} \\
&= -\frac{1}{p}\sum_{k=1}^{p}\sum_{j=1}^{n}\sum_{i=1}^{m}\left(\frac{\partial E^{(2)}}{\partial z_i^{(2)}}\frac{\partial z_i^{(2)}}{\partial\sigma}\right)\sigma'\left(z_j^{(1)}\right)a_k^{(0)} \\
&= -\frac{1}{p}\sum_{k=1}^{p}\sum_{j=1}^{n}\left(\sum_{i=1}^{m}\delta_i^{(2)}w_{ij}^{(2)}\right)\sigma'\left(z_j^{(1)}\right)a_k^{(0)}
\end{aligned}$$

In a similar way, we can compute the partial derivative for the bias as.

$$\frac{\partial E^{(1)}}{\partial b_{ij}^{(1)}} = \frac{\partial E^{(1)}}{\partial z_i^{(1)}}\frac{\partial z_i^{(1)}}{\partial b_{ij}^{(1)}} = \frac{\partial E^{(1)}}{\partial z_i^{(1)}}(1) = \delta_i^{(1)} = \frac{1}{n}\sum_{k=1}^{p}\sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(\delta_i^{(2)}\,w_{ij}^{(2)}\right)\right)\sigma'\left(z_j^{(1)}\right)$$

6. Prepare the XOR data point

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=100,
                           n_features=4,
                           n_clusters_per_class=1,
                           n_classes=2,
                           random_state=seed)

X_train, X_test,
y_train, y_test = train_test_split(X, y,
                                   test_size=0.33,
                                   random_state=seed)
```

In this task, I used sklearn library to generate 100 rows of the classification dataset with 2 classes and each row consisted of 4 features. Then the dataset was spitted into training set and test set with 69% and 33%, respectively.

7. Show how the error changes by iterations.

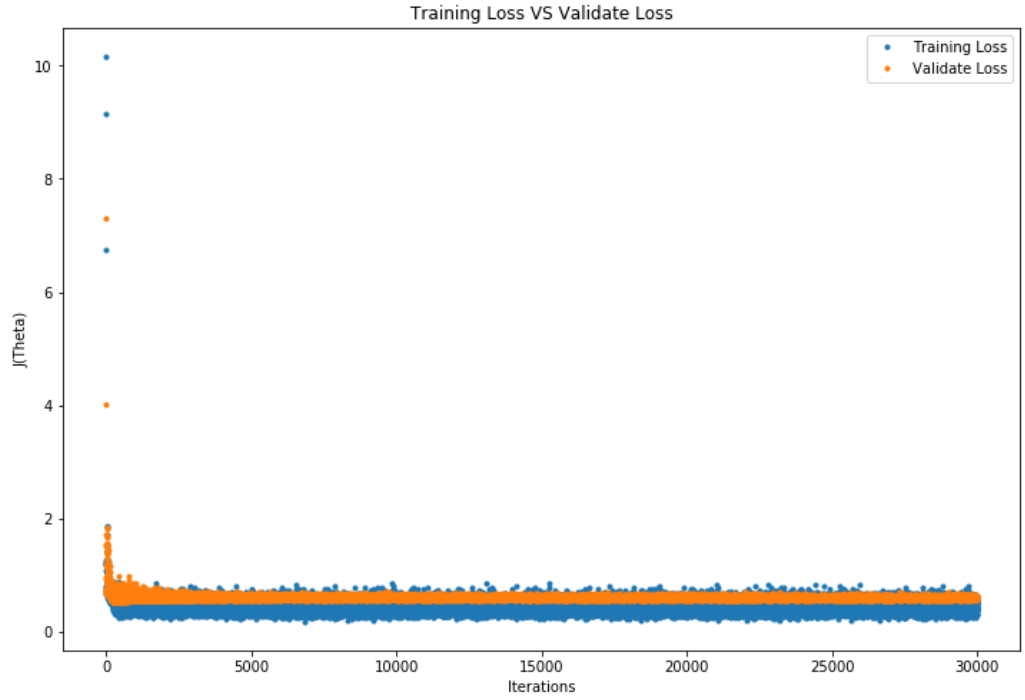

Figure 2 Training Loss VS Validate Loss during training phase

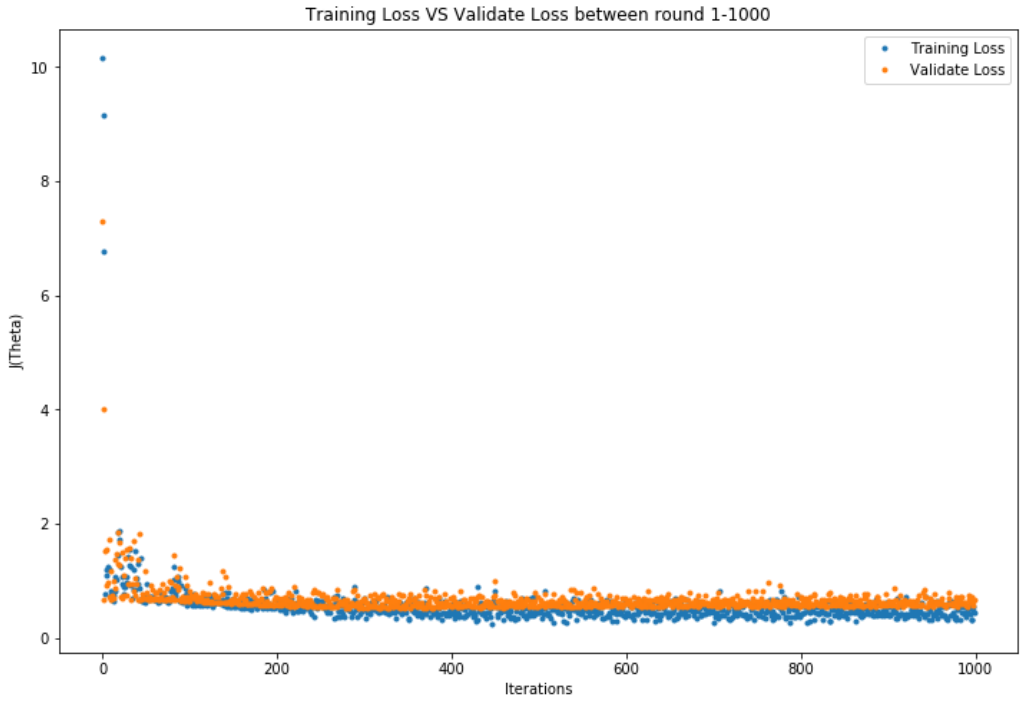

Figure 3 Training Loss VS Validate Loss during training phase between round 1-1000

In Figure 2 and Figure 3, The graph shows the error changes by iterations between the training loss (blue point) and the validate loss (orange point). The graph shows that the error was huge at the beginning (because we randomly initialize parameters) of the training and rapidly reduce and remained stable with a little bit swing because of a nature of the stochastic gradient optimization. The validation loss doesn't seem increase in the long run so the model isn't overfitting.

8. Demonstrate that the network has finally learned the XOR function.

Learned XOR

```
model.W
```

```
[array([[ 0.18972778,  0.68430254, -0.2221906 , -0.11765492],
        [-0.01842504, -0.60790085,  0.38248867,  0.53650324]]),
 array([[-0.32311967,  0.28322874]])]
```

```
model.B
```

```
[array([[1.3295216 ],
        [0.57808959]]),
 array([[-0.22603712]])]
```

```
test_set = load('./test_set.sav')
X_test = test_set[0]
Y_test = test_set[1]
```

```
_, accuracy = model.evaluate(X_test, Y_test)

print("Prediction accuracy: {}".format(accuracy))
```

```
Prediction accuracy: 0.75
```

After the training done, the weight and bias can be used to map any data points to predict the output of the data. The test set was prepared before the training phase and it was evaluated. The result shown that the model can correctly predict 75% of the test set.